Generators in Athena release 6.5.0 and later Updated 23 July 2003

Ian Hinchliffe (I_Hinchliffe@lbl.gov)
Georgios Stavropoulos (George.Stavropoulos@cern.ch)

July 25, 2003

1 Introduction

The individual Generators are run from inside Athena and their output is converted into a common format by mapping into HepMC. A container of these is placed into the transient event store under Storegate. This is presented for downstream use by simulation, for example by the fast simulation. The user is assumed to know how to run athena. The bare minumum is how to make a TestRelease area from which to work. If you do not know this start by consulting the Athena documentation [1]. Once a TestRelease has been set up the sammple jobOptions files.

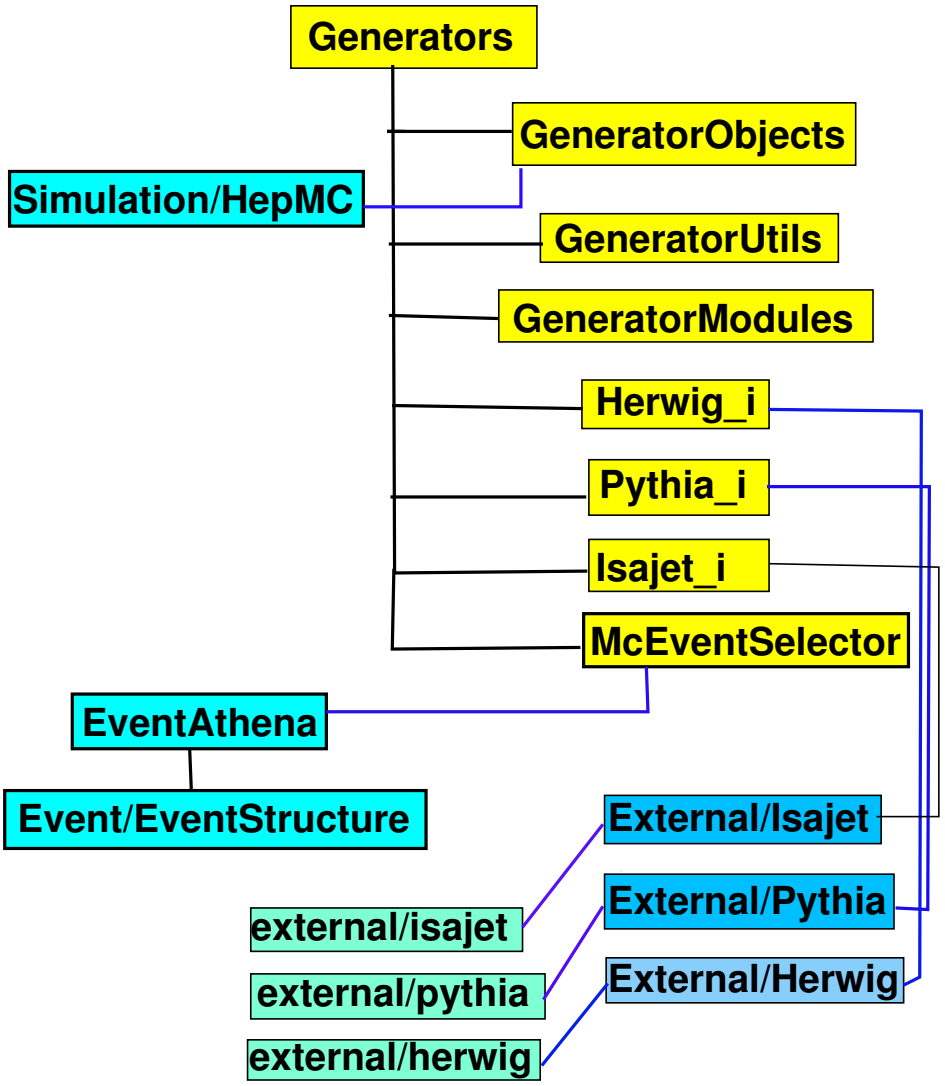
This note describes the overall structure. The user will need to be concerned with GeneratorModules, GenzModule the specific inerface pagkages for each generator such as Herwig_i GenAnalysisTools and GeneratorFilters packages. Each available Generator has seperate documentation describing its use in detail; these are contained in Herwig_i Isajet_i, CompHep_i and these should be consulted. The current list of supported Generators is Herwig, Pythia, Isajet, Hijing, AcerMC, CompHep, AlpGen, Taoula. Photos, Phojet and ParticleGenerator.

The organisation of the code is as follows

- GeneratorModules contains the base classes from which the specific inherit.
- Pythia_i contans the code for the Pythia interface. and the Algrorithm to load Pythia
- Herwig_i contans the code for the Herwig interface. and the Algrorithm to load Herwig
- Isajet_i contans the code for the Isajet interface. and the Algrorithm to load Isajet
- Hijing_i contans the code for the Hijing interface. and the Algrorithm to load Hijing
- Tauola_i contans the code for the Tauola interface. and the Algrorithm to load Tauola
- Photos_i contans the code for the Photos interface. and the Algrorithm to load Photos
- AlpGen_i contans the code for the AlpGen interface. and the Algrorithm to load AlpGen
- Phojet_i contans the code for the Phojet interface. and the Algrorithm to load Phojet
- ParticleGenerator contans the code for the ParticleGenerator interface. and the Algrorithm to load PartilceGenerator

- CompHep_i contans the code for the CompHep interface. and the Algrorithm to load it
- AcerMC_i contans the code for the AcerMB interface. and the Algrorithm to load it
- GeneratorUtils contains some utility routines.
- GeneratorFilters contains some examples of how to filter events.
- GenzModule provides the ability to read events made by the G3 Simulation and pass the envents into Athena in a uniform manner; for example, so they can be used by Atlfast for example
- GeneratorObjectsRoot is a package that outputs and inputs the events in Root I/O format.
- GeneratorObjects sets up the containers which will hold the events in a collection of HepMC events. McEvent also sets up the serialisers for ROOT.
- McEventSelector is responsible for assigning run numbers and providing the hooks to Gaudi-Interfaces, it uses EventAthena.
- GenAnalysisTools is a set of algorthms for Generator analysis, it is used mainly by reconstruction. It contains three packages
  1. CBNT_Truth, used for the truth part of the CBNT combined Ntuple
  2. TruthExamples. This has examples to show histrogramming and listing of generated events
  3. TruthHelper, containing helper classes for extracting stable particles for example.
- PythiaB. Specific version of Pythia used by the B-phyiscs group.

The organisation and dependencies is indicated in the simplified figure.



There are generator specific implementations for a single particle gun, Herwig, Isajet, Tauola, Genz and Pythia. The code for the Generators is in afs/cern.ch/atlas/offline and is linked via linksets defined by External/Pythia, External/Isajet External/Herwig, and External/Stdhep. Detailed documentation on the specific interfaces for the generators can be found in the /doc area of each of the packages listed above.

The HepMC/GenEvent class allows to store into its *signal_process_id* an integer ID that uniquely specifies this signal process. In Generators we allow the use of several generators and it would be usefull to store in this ID the generators combination which was used to produce the event.

For example someone used AlpGen to produce some events, and then Herwig to hadronize them and Tauola to decay the taus. This requires to adopt a certain convention for this *GenEvent/signal_process_id*. The one adopted is

$$signal\_process\_id = I * 1000000 + J * 100000 + K * 10000 + process$$

where

I: 1 = Pythia, 2 = Herwig, 3 = Isajet, 4 = Single, 5 = Hijing, 6 = Phojet
J: 1 = Comphep, 2 = User, 3 = Acermc, 4 = Alpgen
K: 1 = Tauola, 2 = Photos, 3 = TauolaAndPhotos

So, in the example above the ID will be
$signal\_process\_id = 2 * 1000000 + 3 * 100000 + 1 * 10000 + Alpgen\ process$

This convention is implemented into the Generators/GeneratorModules/GeneratorName as an enum. There somebody can find also several methods to unpack the *signal_process_id*.

2 GeneratorFilters

This package contains some very simple examples of how to filter generated events. A base class (GenFilter) is provided to open the event collection. The actual filters inherit from this class. Examples are provided ElectronFilter LeptonFilter and ZtoLeptonFilter. The first two pass events that have either an electron or a lepton in the specified P_T and η ranges (these can be set from jobOptions). The last passes events that have a Z decaying to leptons.

The filter should be used in a **Sequence** as follows.

```
ApplicationMgr.DLLs  += { "Herwig_i","TruthHelpers" };
ApplicationMgr.DLLs += { "HbookCnv" };
ApplicationMgr.DLLs += { "GeneratorFilters" };
ApplicationMgr.DLLs += {"GaudiAlg"};
ApplicationMgr.HistogramPersistency = "HBOOK";
ApplicationMgr.TopAlg = {"Sequencer/Generator"};
Generator.Members = {"Herwig", "ZtoLeptonFilter", "HistSample"};
HistogramPersistencySvc.OutputFile  = "herwig.hbook";
NTupleSvc.Output     = { "FILE1 DATAFILE='herwigtuple1.hbook'
OPT='NEW'" };
```

This will call Herwig and any events that have a $Z \rightarrow leptons$ in them will pass the filter and be processed by the HistSample Algorithm. If an event fails the filter it is thrown away and the sequence restarts.

References

[1] http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/index.html